POLITECNICO DI TORINO
Repository ISTITUZIONALE

Performance comparison and optimization of ICN prototypes

Original
Performance comparison and optimization of ICN prototypes / SAFARI KHATOUNI, Ali; Mellia, Marco; Venturini, Luca; Gallo, Massimo; Perino, Diego. - ELETTRONICO. - (2016). (Intervento presentato al convegno Information Centric Networking Solutions for Real World Applications Workshop co-located with the 2016 IEEE Global Communications Conference tenutosi a Washington DC (USA) nel December 8, 2016) [10.1109/GLOCOMW.2016.7848997].

Availability:
This version is available at: 11583/2653502 since: 2017-03-19T23:26:17Z

Publisher:
IEEE

Published
DOI:10.1109/GLOCOMW.2016.7848997

Terms of use:
openAccess

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

04 May 2023

Performance Comparison and Optimization of ICN Prototypes

Ali Safari Khatouni[1], Marco Mellia[1], Luca Venturini[1], Diego Perino[2], Massimo Gallo[2]
[1]Politecnico di Torino, Italy - first.last@polito.it
[2]Bell Labs, Nokia, France - first.last@nokia.com

***Abstract*— Information Centric Networking paradigm is a solid proposal to update the original Internet design. While the research community is still working on technical improvement, the technology has reached a stage that allows the design, testing and deployment of ICN specific applications. To this goal, the development of testbeds allowing researchers to evaluate their applications in a real setup is of key importance. There are few existing ICN prototypes that would allow experimenters to work in this direction. Their performance is fundamental, so that they do not create artificial bottlenecks affecting test reliability. Unfortunately, existing solutions have been designed to demonstrate the feasibility of the ICN solution, thus lacking of optimization and scalability required to run reliable scalability tests. In this paper we address this problem by first, offering a performance comparison between available ICN prototypes and, second by proposing a revisited system design, CCN-par, to increase scalability, and remove bottlenecks that hamper current software-based testbeds.**

I. INTRODUCTION

After about 10 years of research, ICN paradigm has reached a sufficient level of maturity and researchers are now designing and testing applications specifically designed for it. Since the very early stage, prototypes have been made available to let researchers experiment with the new technology. Two of the most popular solutions are the CCNx by the Palo Alto Research Center (PARC) [1], and the CCN-lite [2], a lightweight and functionally compatible implementation of CCNx. Both provide all the support to create a ICN-enabled testbed, and are commonly used to demonstrate ICN protocol and application feasibility. Besides those two, the NFD [3] prototype has also been released by the NDN project and is currently used in the NDN test-bed. Nodes in ICN play an important role, since they have to support and manage the forwarding operations, which are based on content names instead of content locations (i.e., IP addresses). Each node has to keep the Pending Interest Table (PIT), where the status of forwarded content requests is maintained, the Forwarding Information Base (FIB), where forwarding information is stored, and the Content Store (CS), i.e., the local cache of contents that can be served locally. These new elements call for specific and optimized data structures, and their intertwining may create bottlenecks that emerge when stressing performance. CCNx, CCN-lite and NFD however have been designed mostly for demonstrating the feasibility of the approach, and they lack the optimization that is needed when one would like to stress the overall performance of the system. Hardware solutions would be the natural choice for researchers that would like to test application performance without incurring into testbed scalability issues. Perino et al. [1] and So et al. [2], designed high-end content routers that can support high speed name-based forwarding in hardware. However, the cost of these solution is very high, and software-based prototypes are preferred and commonly used for ICN application scalability testing. The three aforementioned prototypes implement two different, but very similar, ICN proposals, namely CCN (CCNx, CCN-lite) and NDN (NFD). CCN and NDN share most of the architecture design except for name-lookup algorithms used in PIT and CS. Indeed the first (CCN) performs longest prefix match on the FIB and exact match on PIT and CS. On the contrary NDN uses longest prefix match for all of its data structure (i.e., FIB; PIT, CS) allowing content consumers to request desired piece of data for which they do not know (completely) the name. While this feature is very interesting it can potentially prevent testbed scalability targeted in this work. For this reason, in the following, we focus on CCN.

In this paper we present a performance comparison of CCNx and CCN-lite. Based on our results, we identify bottlenecks and present CCN-par (as Paris, being it developed during an internship in Paris), an optimized version of CCN-lite in which all critical data structures (of PIT, FIB, and CS) have been optimized by using hash tables instead of simple but inefficient lists, and by optimizing the queuing disciplines that packets have to follow inside the Linux kernel. In addition, we add support to the new dynamic forwarding paradigm proposed by Carofiglio et al. [3]. CCN-par is tested and its performance compared against CCNx and CCN-lite under different scenarios. We use the Grid-5000 distributed testbed, in which Linux nodes in France cities are connected by 10Gbps WAN links. Experiments show that CCN-par guarantees consistent scalable performance, topping to more than 50,000 packets per second of forwarding rate, i.e., 2.5x faster than CCNx and 5x faster than CCN-lite. In a nutshell, CCN-par allows researchers to test their application in a system where the testbed is not the bottleneck per se, and thus to evaluate the performance of the application in a consistent and coherent way. CCN-par is currently proprietary design of Nokia Bell Labs, and we plan to open it for researchers in the near future.

II. SYSTEM DESIGN

In this section we describe CCN-par, a Linux kernel module written in *C* implementing a forwarding node of the CCN architecture. It can be installed on Linux kernel as a kernel module and is backward compatible with CCNx and CCN-lite. CCN-par is based on CCN-lite, which uses simple linked list for implementing CCN node basic building blocks and hence is a good starting point for building a scalable CCN prototype. Thanks to its modularity CCN-lite (and hence CCN-par) allows a gradual optimization of its basic components and the possibility to evaluate the gain at each step. The kernel module is built on top of the NaNET socket designed by Gallo et al. [4].

[1]http://blogs.parc.com/ccnx/
[2]http://www.ccn-lite.net/
[3]http://named-data.net/doc/NFD/current/

A. Data Structure

In CCN nodes data structures have a direct impact on the performance of basic building blocks. The FIB has been deeply investigated to be efficiently implemented by Tree Bitmap, Bloom Filter, or Cookoo Filter in [5], [6], [7], [8], [9], [10], [1] and [11] respectively. There are several known structures which are used for the implementation of the PIT: Counting Bloom Filter [12], [1], Hash-Table [13], [14], [15], and Name prefix trie [16]. These sophisticated structure can be adopted in the other building blocks too. However, CCN-lite uses a linear list for all data structures for the sake of simplicity. In CCN-par we replace linked lists with a Linear Hash Table (LHT) with the twofold objective of maintaining reasonable complexity and providing efficient lookup i.e., $O(1)$, despite the fact that it is not the most efficient data structure in terms of memory usage. We plan to extend our work by verifying the main bottleneck of our kernel module and improve it with more efficient structures.

B. Scheduling

Our kernel module runs in a single thread. Hence, all packets are processed in the main thread. If a packet arrives during the process of another one, the module receives the kernel interrupt of the packet but postpones the packet processing. At packet arrival, our kernel module needs to schedule the received packet. We use a non preemptive scheduler, that first finishes the process of the current packet and then starts the process of the next packet in the waiting queue. To this end, we use *Workqueue*, the kernel facility to call a function at future time. There are two main approaches to use Workqueue: using the global Workqueue or creating a dedicated Workqueue for our system. Global Workqueue is the shared Workqueue that is used by the device driver and kernel itself. Our kernel module schedules packets with a Workqueue such that works are simply packets' processing that need access to CPU. Therefore, we define a dedicated Workqueue to put all arriving packets' processes work inside Workqueue. The kernel will then put a packet under service after finishing the previous works (packets in queue). CCN-lite uses a global Workqueue that, as we show in the Sec. IV, causes a kernel crash because the module interferes with the kernel I/O.

C. Aging Mechanism

The content consumer is responsible for requesting the content by mean of generating an interest packet, if he/she wants the Data. Interest and data packet in the network are associated with expiration times (i.e., PIT timer for the interest and content validity for the data). Expiration time is useful to create soft state system and protect CCN node from different kinds of security attacks. Occasionally, malicious users may try to create an interest flooding attack by requesting multiple times non existing contents. PIT timers are also essential in case of packet losses, in order to delete the corresponding PIT entry and allow a re-expressed interest to be forwarded. Accordingly, aging is implemented for all CCN-par building blocks to protect them from different types of attacks and keep valid data in tables. Each building block's entry has a different expiration time and periodically, the system checks entries in all tables to find expired entries. The potential drawback is that at each check, we spend CPU time and lock the table under control. To avoid this, and schedule the aging operation

Fig. 1: CCN-lite Workqueue.

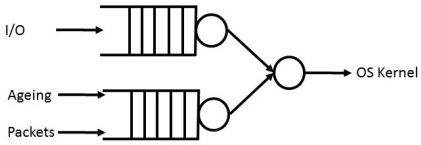


Fig. 2: CCN-par Workqueue.

in the future, we create a Workqueue which, as soon as an entry is inserted to the table, stores the corresponding work in charge of the deletion of the expired entry. With the modified design, the Linux kernel will invoke the deletion function in a future time equal to the expiration time of the inserted entry. Figs. 1, 2 illustrate the queue design in CCN-lite and CCN-par respectively. Fig. 2 presents the CCN-par dedicated Workqueue for aging and scheduling and the global Workqueue used by the host OS.

D. Cache Replacement Policy

The built-in caching capability of CCN provides effective content distribution at a global scale without requiring special infrastructure. Therefore, we replace the linked list of CCN-lite with LHT to organize CS. The important part in the cache design is the replacement policy. Replacement policy can affect the performance of the node because if the node does not store popular contents, cache miss will occur, causing an additional lookup in the FIB, the interest will be eventually forwarded, and finally satisfied with the permanent copy of the content. Least Recently Used (LRU) has been used in the context of CCN [17], [18], [19]. Other works consider different policies, such as Most Recently Used (MRU) and Most Frequently Used (MFU). Interestingly, [20] shows LRU performance to be indiscernible from MRU/MFU, but much better than the single FIFO offered by CCN-lite. In CCN-par, we implement LRU through a list of recently used entries that is directly connected with the LHT for fast look-up, insert, and delete operations.

E. Dynamic Forwarding Algorithm

Carofiglio et al. [3] propose a family of optimal distributed algorithms for interest forwarding. The algorithm is optimal as it minimizes the total number of pending interest at the node. In CCN architecture, each node stores the number of unsatisfied interests in the PIT. The number of pending interests reflects (i) content proximity: The length of the path and the response time associated to the given content; (ii) Congestion status. As observed in Sec. I, the interest forwarding decision is based on the LPM to find the output interfaces in the FIB. The CCN node collects the data from the network which are intended to handle packet from input to the potential sources. Moreover, it is possible to exploit them to provide congestion control mechanism and multi-path forwarding. We use the algorithm designed in [3]. Let us describe the main steps of the algorithm. At each packet reception (Data or Interest), the number of Pending Interest (PI) which associate to a given FIB entry and a particular output interface is updated, this update may be incremental or decremental. The interface average is recomputed with the instantaneous value of PI and the moving

average is updated repeatedly itself. The output interface selection is based on the random weighted algorithm for each arriving packet. The weight normalization is done according to the normalized weights for each prefix/output interface and sum of all weights of all interfaces of the particular prefix. At the beginning of the algorithm each interface has a weight equal to one. Accordingly, after node startup, the forwarding process is uniform over all available output interfaces.

III. Experiment Setup

Our experiments were conducted in Grid5000 [21]. Grid5000 is an experimental platform distributed over different sites across France, offering a monitoring infrastructure that makes evaluating a new network design simple and efficient. Grid5000 is a perfect tool to create an isolated network to test network applications and distributed computing and obtain results that are not affected by high CPU load caused by concurrent experiments thanks to the exclusive reservation of the content producers available. High speed links assure negligible and constant latencies between the different nodes, and avoid bandwidth bottlenecks that are important in our test because we want to be sure to have the bottleneck on CCN-par. For our tests, we built a custom Linux kernel image (Debian Ubuntu 3.2.0-24-generic x86_64 GNU/Linux), including our kernel module, CCNx software, and several other applications we need to analyze the results and used it as the operating system that is deployed on reserved grid5000 nodes. We use a set of machines located on the Nancy site equipped with dual 2.5 GHz Intel Xeon L5420 Quad-core and 15GB RAM.

To orchestrate our tests we used a modified version of Lurch [4], an open source tool that allows a complete setup of a CCN network and the execution of some automated tests, together with statistics collection. Lurch builds a topology, by limiting the link capacity between nodes by means of token bucket regulators, sets up the routing tables and the CCN faces and sets up the content repositories (i.e., content producers). When running a test, Lurch run content producers and content consumers, generating random requests according to different distributions and file popularities, collect the statistics from network interfaces, cache and CPU and gather all the logs in an archive. The CCNx implementation evaluated in this work is CCNx-0.8.0 wile CCN-lite is 0.2.0. Lurch starts ccncatchunks2, an application on top the CCNx node, used to retrieve named contents, at each content consumer node in our topology. The ccncatchuck2 application generates interest packets with a specific content name and a sequential chunks number. In the topology used in the experiment some nodes are designed as content producers and run ccndelphi, an application available in lurch, that replies randomly generated content to interests belonging to a specific prefix. The ccndelphi application available in Lurch runs on content producers and replies with random generated content to every interest that it receives with specified name prefix.

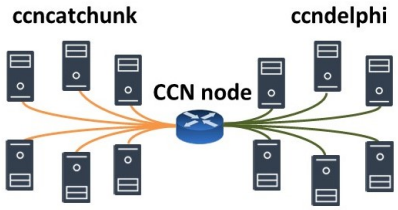
A. Performance metrics

We define several metrics according to the nature of the CCN router. Additionally, we collect the processing time of each building block of CCN-par. Metrics are used to evaluate the efficiency of the prototypes, including the time spent for different operations in CCN-par. These metrics are described in the following:

- Packet forwarding rate: Routers should be designed and tuned to be robust against the worst case scenario. In particular, they should be able to forward at peak packet rate. The main performance metric in our analysis is peak and average packet forwarding rate of the prototypes.
- Bandwidth usage: Content consumers need to retrieve contents which split in different chunks. An important factor to monitor is the link bandwidth usage. In fact nodes contribute to deliver chunks from content producers to the consumers. Hence, one of the important goal of the network is minimizing its total bandwidth usage.
- Average packet processing time: To better understand the actual relevance of the various operations. We need a deep analysis of the processing time of a packet inside a CCN-par node. To this goal, we define particular metrics for the monitoring of packet processing in relay node. We collect quantitative data about crucial operations of the node. We consider average packet processing time, content processing, interest processing, packet parsing, nonce lookup, FIB lookup, and PIT lookup.

[4]http://systemx.enst.fr/lurch

Fig. 3: An example of 6-1-6 star topology for experiments.



IV. Results

In this section, we present our performance evaluation of CCN-par, CCN-lite, and CCNx. We designed the experimental scenarios in order to highlight the behavior of a CCN router in working conditions. Specifically, in IV-A we analyze a scenario where a router is pushed to reach the peak of its processing capabilities; in IV-B we explore how much the same router is relieved by introducing the caching capability; and, finally, in IV-C we show how a CCN node reacts to sudden link failures.

A. Packet Forwarding Without Cache

We assume the peak throughput is obtained when the CPU of the router is fully utilized. In this test, content consumers send interest packets to the content producers via the CCN router, and then the content producer sends back corresponding data packets via the CCN router. To saturate the CCN router, we use several content consumers to generate interest packets, and multiple content producers to respond with data packets. Fig. 3 shows the network configuration for the first scenario. Each machine in the topology represents a dedicated machine as described in Sec. III.

In this set of experiments, the size of CS, which serves as cache for the contents, is set to zero to eliminate the content

processing time for CS. The names are distinct and contain three components and the length is variable from 15 characters to 35 (e.g., ccnx:/nokia/OBJNUM0000/1234). All tables have equal size (15,000 elements) except the CS. The workload is generated by content consumers at Constant Bit Rate until the link is saturated. The test is repeated with different number of nodes from 7 to 51 nodes in order to increase the load on the router. Every content consumer runs the ccncatchunks2 to send interest packets. Interests have names like $ccnx : /prefix/OBJNUM0000/chunk\_number$, where prefix has 5 to 15 characters and $chunk\_number$ is the progressive identifier of the chunk. Interest packets sent from content consumer ith will be routed to content producer ith, which runs the ccndelphi program to reply with randomly generated data packets. The data packet is set to 1024 Byte.

We measure the packet forwarding rate of the CCN-par, CCNx, and CCN-lite as shown in Fig. 4. Each point represents an average value over 5 repetitions. Y-axis shows the packet forwarding rate in Packet per Second (pps) and X-axis indicates the topology of the network, for instance 3.1.3 indicates 3 content consumer-producer pairs and the node under test in the center of the network. Fig. 4 illustrates that CCN-lite cannot

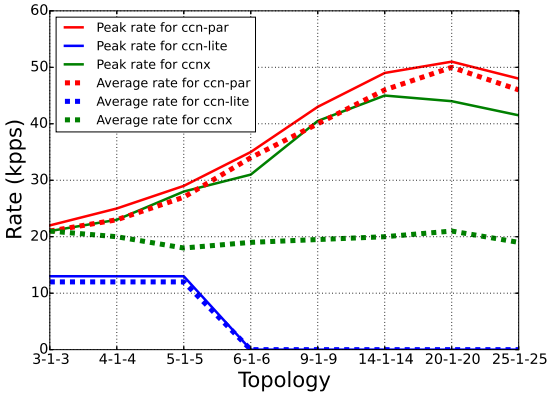


Fig. 4: Packet forwarding rate.

forward packet at a rate higher than 13 kpps (considering the experiment setup at Sec. III). Moreover, whenever the rate goes higher, the kernel abruptly crashes. From our investigations, the main reason of the crash is the scheduling mechanism, as it uses a system-global workqueue to schedule the processing of the arriving packets. The same workqueue is used by Linux to schedule I/O operations, which are thus interfered by CCN-lite, consequently resulting in a kernel crash.

Both CCNx and CCN-par sustain higher peak rates than CCN-lite, however Fig. 4 shows a considerable difference between the average and the peak rate of the CCNx (continuous-line and dashed-line respectively). The main reason of this low average performance can be found in a very unstable behavior, much fluctuating between highs and lows, whereas CCN-par has a way steadier forwarding rate. To the best of our knowledge, the main reason of this behavior is the CCNx aging process and scheduling. Indeed, the aging mechanism in CCNx periodically probes all expired elements slowing down the node, using CPU resources and block tables shared with the other CCNx processes.

In Fig. 5 and 6, Y-axes and X-axes illustrate CDF and the transmission rate in KByte/sec (KB/s) respectively. Solid

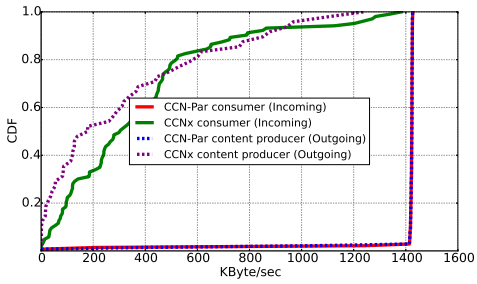


Fig. 5: Results for nodes without cache on data packet direction.

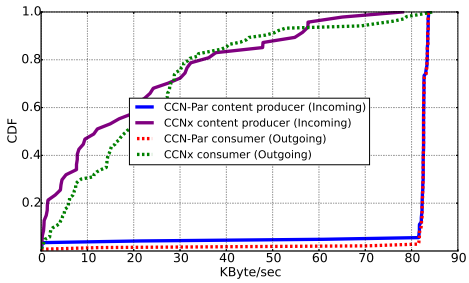


Fig. 6: Results for nodes without cache on interest packet direction.

FIB Lookup	Packet parsing	Nonce Lookup	PIT Lookup
2800 ns	2000 ns	850 ns	180 ns

TABLE I: Average Tables Lookup.

and dashed-line in Fig. 5 show the transmission rate for the data packet direction on the link between central node and a consumer or a content producer respectively. In our terminology, incoming means all packets (data and interest) received by a node and outgoing indicates all packets are sent by a node. It is worth noticing that the CCN-par has a stable forwarding bit rate at 1400 KByte/s when there are interests from the content consumer. Notice that the rate of CCNx is really low in average and unstable. These results illustrate that CCN-par supports more stable bit rate. Figs. 5 and 6 show that the captured traffic on the central node has almost constant bit rate. In terms of bandwidth usage the links between the producer and the CCN router and the links between the content consumer and the CCN router exhibit the same bit rate because the node does not drop packets and does not respond with cached content.

Table I presents a detailed view on the average packet processing time in CCN-par. For each packet either interest or content computes the processing time in each main building blocks. Moreover, to have more accurate value each sample is an average of thousand packets, the Table I illustrates an average of million samples. Table I indicates that the FIB lookup time is less than 3000 $nano$ seconds and it is triple as nonce lookup time. The FIB lookup is the most expensive operation in CCN-par because we use LPM to forward packet to the destination and we implement dynamic forwarding mechanism on FIB which causes extra operations. Moreover, the reason to have costly FIB lookup in our test is that the FIB entry has name prefix as ccnx:/nokia but the interest name has three components like ccnx:/nokia/OBJNUM0000/1. Accordingly, the LPM in FIB needs three lookup operations for name prefix size to find the output face. The PIT lookup

spends 180 *nano* seconds and is significantly lower than the other operations. At first glance it seems odd but there are two main reasons. First, we use exact match which is faster than LPM. Second, ccncatchunks2 generates a new interest after the receiving of the last interest or expiration time of the last interest. Therefore, in our network we do not have many unsatisfied interest packets in the PIT table. According to the result of the other building blocks, if the PIT table is full, we observe a PIT lookup time of about 3μ, smaller than the one for the FIB. The second most expensive operation is packet parsing. It uses 2μ seconds and it is more costly than nonce lookup. This high cost for parsing is mainly due to the choice of using TLV schema for the packet, header fields instead of fixed-length. The TLV indeed allows for customizations of the protocol, but results in much longer packet processing time, as highlighted in Table I.

B. Packet Forwarding With Cache

In this scenario, we show the results for the node with cache. Consumers send interest packets with content name as defined in Sec. IV-A to the central node, which replies with content in cache if present or forwards it to the possible content producer(s). We choose the topology with 20 content consumer-producer pairs as presented in Fig. 3. The popularity of the contents in the Internet is the critical factor which influences the behavior of caches. As a matter of fact, it is well known that the most of the traffic is made by the few number of popular contents, while the few part of the demand is made by almost single requests for very rare contents. In terms of probability distributions, such a behavior is usually associated to the $Zipf$ distribution. Therefore, we used $Zipf$ distribution for popularity of content. In this set of experiments, we set the CS size equal to 15,000 contents.

The first set of results indicates that the peak forwarding rate is almost equal to the results in Sec. IV-A but the average forwarding rate is 10% less (is not reported here because of lack of space). However, we should take into account that we can compare these two scenarios in terms of network resource usage. Delivering a single data packet from a content producer to the central router would result in satisfying future interest packets for the same data packet. Since data packets are larger than interest packets, the outgoing data rate should be much higher than incoming data rate, as depicted in Fig. 7.

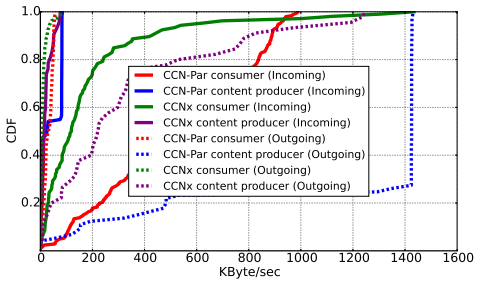


Fig. 7: Results for nodes with cache.

Fig. 7 demonstrates that serving content directly from CS is more efficient and improves network throughput. Fig. 7 shows that in case of CCN-par, the link in producer side experiences less traffic at peak with respect to consumer side. In other words, interests are satisfied by the CS in CCN-par. Table II details the average processing time for different type of packets

Content	Interest	All
43 µs	5 µs	21 µs

TABLE II: Average packet processing time

in CCN-par. Unexpectedly, the average content processing time is significantly higher than interest processing time. Content packet forwarding has an average time equal to 43μ seconds. Although, it is two folds of the average time spent for all packets. Remember that in a CCN node contents are stored in a cache based on their cache policy. Hence, the main reason of this behavior is that for each data packet in the current scenario, we have extra operations: first, we have a lookup in cache; if it exists, it will be dropped. Otherwise, it will be added to CS and the LRU should be updated. The CS becomes full after few seconds of run. Therefore, each insertion in CS needs a deletion in advance.

C. Dynamic forwarding

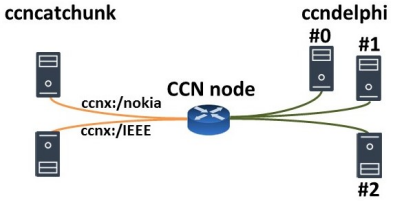
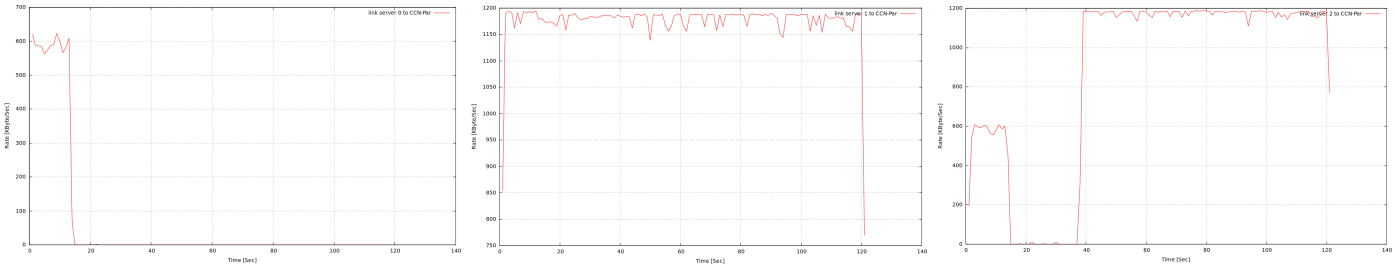


Fig. 8: Network topology.

We focus on a simple multi-path network scenario, where users are connected to three repositories via three non disjoint paths. Usersaccess the central router and retrieve two different named objects with a prefix as we mentioned in Sec. IV-A. We define a toy network to show the effect of the dynamic forwarding in CCN-par. In this topology we can easily visualize in details what happens to specific name prefix with different output faces in FIB. Fig. 8 shows the network topology and the role of each node. As depicted, consumers request two different name prefix. Moreover, the relay node forwards interest packets to three distinct repositories, the name prefix (ccnx:/nokia) has two output faces in the relay node. In this set of experiments, we set the size of CS equal to zero to prevent the CS processing time. All tables have size equal to 15,000 element. Notice that the workload is generated by content consumers at CBR until the link is saturated. Request forwarding decisions are based on the selection of LPM interfaces in the FIB. Indeed, FIB entries specify name prefix rather than full object names because it is unfeasible to maintain per-object name information. At each packet reception, the value associated to a given name prefix and to a particular output interface is updated. At the beginning, each interface has a same weight and the packet forwards uniformly over available output faces.

We now focus on the bit rate of the links from content producers to the central node. As shown in Fig. 9 the traffic from producer #1 to CCN-par is steady at 1200 KByte/Sec because there is just one output face for this name prefix. However, the second prefix (ccnx:/nokia) has two available faces in FIB. The producer #0 is turned off after 15 seconds to see how the dynamic forwarding algorithm reacts. At the beginning of the test, the interest are forwarded uniformly between the two available output faces. When content producer #0 stops, the content consumer still waits for receiving the last requested contents, that content producer #0 will never provide. In the



(a) Link between content producer #0 and CCN-par.

(b) Link between content producer #1 and CCN-par.

(c) Link between content producer #2 and CCN-par.

Fig. 9: Results for dynamic forwarding.

meanwhile, CCNx deamon retransmits unsatisfied interests, which are discarded by the router since the first interest still exists in the PIT. The node deletes the PIT entry after a timeout of 20s and all interest packets are forwarded to the only active content producer for the same name prefix.

V. Conclusion

CCN is a state-of-the-art communications technology. It holds an important promise to enable new services and create a new category of applications software. Our work explores the implications of CCN on software router design, storage and network performance. Our CCN architecture and implementation operates in a wide range of network environments. To assess the performance of the design, we have implemented the proposed prototype based on CCNx protocol and set up a testbed for large scale experimentation. The experimental evaluation in various network scenarios confirms efficiency and robustness w.r.t the other existing prototypes. We evaluate the viability of building a high-performance CCN router out of common PC hardware and the Linux system. Our preliminary evaluation shows that CCN-par supports forwarding at 50 kpps.

There are few research challenges that need to be solved before applying CCN in today's Internet. First, we need to design data structures which can support name forwarding and lookup at wire-speed. For instance, designing a high-performance distributed algorithm for FIB lookup. Second, CCN architecture has been proposed recently by researchers, which means that it is still at an immature step. For the future work we design the FIB with a more efficient data structure as mentioned in Sec. II. In addition, packet parsing is the second costly operation in our design which can be improved by means of a more sophisticated data structure or design well structured data packet format instead of the current TLV version.

References

[1] M. Varvello, D. Perino, and J. Esteban, "Caesar: A content router for high speed forwarding," in *Proceedings of the Second Edition of the ICN Workshop on Information-centric Networking*.

[2] W. So, A. Narayanan, and D. Oran, "Named data networking on a router: Fast and dos-resistant forwarding with hash tables," in *Proceedings of the Ninth ACM/IEEE ANCS*.

[3] G. Carofiglio, M. Gallo, L. Muscariello, M. Papalini, and S. Wang, "Optimal multipath congestion control and request forwarding in information-centric networks," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*.

[4] M. Gallo, L. Gu, D. Perino, and M. Varvello, "Nanet: Socket api and protocol stack for process-to-content network communication," in *Proceedings of the 1st International Conference on Information-centric Networking*.

[5] W. Eatherton, G. Varghese, and Z. Dittia, "Tree bitmap: Hardware/software ip lookups with incremental updates," *SIGCOMM Comput. Commun. Rev.*

[6] W. Quan, C. Xu, A. Vasilakos, J. Guan, H. Zhang, and L. Grieco, "Tb2f: Tree-bitmap and bloom-filter for a scalable and efficient name lookup in content-centric networking," in *Networking Conference, 2014 IFIP*.

[7] Y. Wang, K. He, H. Dai, W. Meng, J. Jiang, B. Liu, and Y. Chen, "Scalable name lookup in ndn using effective name component encoding," in *Proceedings of the 2012 IEEE 32Nd International Conference on Distributed Computing Systems*.

[8] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor, "Longest prefix matching using bloom filters," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*.

[9] H. Song, F. Hao, M. Kodialam, and T. Lakshman, "Ipv6 lookups using distributed and load balanced bloom filters for 100gbps core router line cards," in *INFOCOM 2009, IEEE*.

[10] S. Ding, Z. Chen, and Z. Liu, "Parallelizing fib lookup in content centric networking," in *Networking and Distributed Computing (ICNDC), 2012 Third International Conference on*.

[11] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*.

[12] W. You, B. Mathieu, P. Truong, J. Peltier, and G. Simon, "Dipit: A distributed bloom-filter based pit table for ccn nodes," in *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*.

[13] D. Perino and M. Varvello, "A reality check for content centric networking," in *Proceedings of the ACM SIGCOMM Workshop on Information-centric Networking*.

[14] H. Yuan, T. Song, and P. Crowley, "Scalable ndn forwarding: Concepts, issues and principles," in *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*.

[15] M. Varvello, D. Perino, and L. Linguaglossa, "On the design and implementation of a wire-speed pending interest table," in *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*.

[16] H. Dai, B. Liu, Y. Chen, and Y. Wang, "On pending interest table in named data networking," in *Proceedings of the Eighth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*.

[17] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Modeling data transfer in content-centric networking," in *Proceedings of the 23rd International Teletraffic Congress*.

[18] L. Muscariello, G. Carofiglio, and M. Gallo, "Bandwidth and storage sharing performance in information centric networking," in *Proceedings of the ACM SIGCOMM Workshop on Information-centric Networking*.

[19] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," *Relatório técnico, Telecom ParisTech*.

[20] K. Katsaros, G. Xylomenos, and G. C. Polyzos, "Multicache: An overlay architecture for information-centric networking," *Comput. Netw.*

[21] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard, "Grid'5000: a large scale and highly reconfigurable grid experimental testbed," in *Grid Computing, 2005. The 6th IEEE/ACM International Workshop*.